



Creación de Aplicaciones Python

**Lección 10: Frameworks GUI [1/2]:
Tkinter**

ÍNDICE

Lección 10. – Frameworks GUI: Tkinter [1/2]	2
Presentación y objetivos	2
1. Tkinter: Conceptos Básicos.....	4
2. Transformar en Aplicación de Escritorio	17
3. PyGubu: Un “Builder” para TKINTER	21
4. Puntos clave.....	29

Lección 10. – Frameworks GUI: Tkinter [1/2]

PRESENTACIÓN Y OBJETIVOS

GUI que viene de *Graphical User Interface* se refiere a aquellas aplicaciones que permiten servir como puente de comunicación entre nosotros y ordenadores, máquinas o dispositivos por ejemplo. De tal manera que lo que son es, como su nombre indica, una Interfaz Gráfica de Usuario.

Existen diferentes Frameworks en Python para este objetivo.

A año 2021, se podría citar los siguientes:

- | PyQT: El cual explicaremos en este curso. (PyQT5/PyQT6)
- | Tkinter: El tema que nos ocupa
- | PyGTK: El cual finalmente no será explicado
- | Kivy: Otro Framework de los favoritos que no será explicado.

Y un largo etc.

Los proyectos prototipo suelen ser:

- | Calculadora para el Ordenador,
- | Editores de Texto,
- | Se puede crear una aplicación para controlar un sistema electrónico,

Como la mejor forma de verlo es programando, veremos paso a paso los conceptos fundamentales del Framework Tkinter (en esta lección) y veremos PyQT (en la siguiente lección).

Trataremos de explicar los conceptos fundamentales de forma que sea fácil de entender, fácil de seguir, y tratando de orientar a el/la alumno/a en futuros proyectos que desarrolle de manera personal o para una compañía.

Lo primero que mencionar de Tkinter es que es un Framework GUI de los favoritos, dado que es fácil de aprender, quizá no tiene el mejor diseño, quizá PyQt es algo mejor, y disponer de “builder”, al igual que PyQt.

Veremos a continuación con explicación práctica.

La web es la siguiente:

<https://docs.python.org/3/library/tkinter.html>

En la misma podemos encontrar información adicional.



Objetivos

- Conocer los algunos Frameworks GUI
- Uso de Tkinter para hacer Aplicaciones
- Conocer algunos “builders” que tenemos

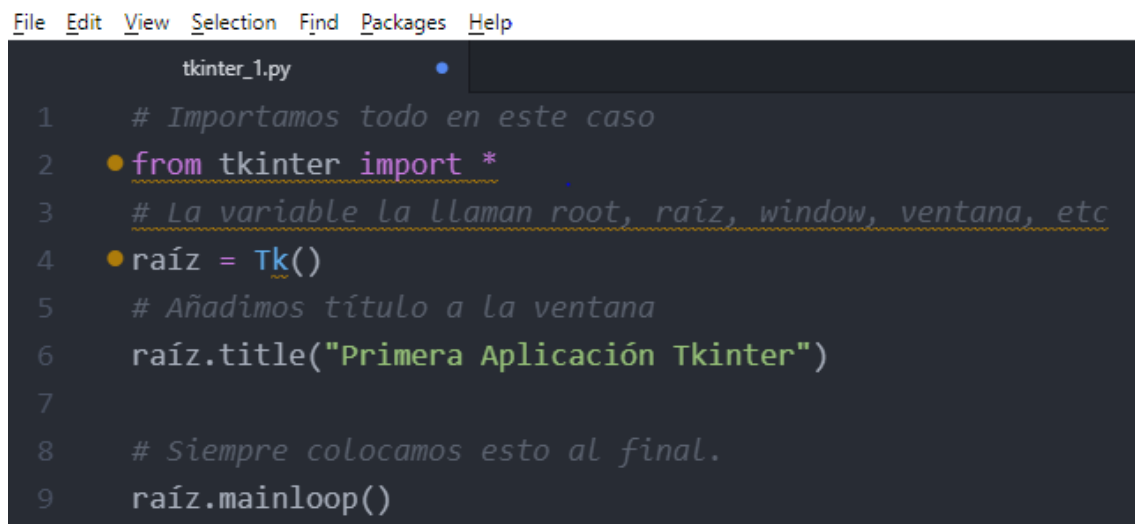
Comencemos !

1. TKINTER: CONCEPTOS BÁSICOS

INSTALACIÓN

LINUX # \$sudo apt-get install python3-tk

aunque viene por defecto normalmente con python

A screenshot of a code editor window titled 'tkinter_1.py'. The editor has a menu bar with 'File', 'Edit', 'View', 'Selection', 'Find', 'Packages', and 'Help'. The code is as follows:

```
1  # Importamos todo en este caso
2  • from tkinter import *
3  # La variable la llaman root, raíz, window, ventana, etc
4  • raíz = Tk()
5  # Añadimos título a la ventana
6  raíz.title("Primera Aplicación Tkinter")
7
8  # Siempre colocamos esto al final.
9  raíz.mainloop()
```

Figura 1.1: Ejemplo básico con Tkinter.

La forma rápida de visualizar la aplicación, mientras comenzamos a programar será la siguiente:

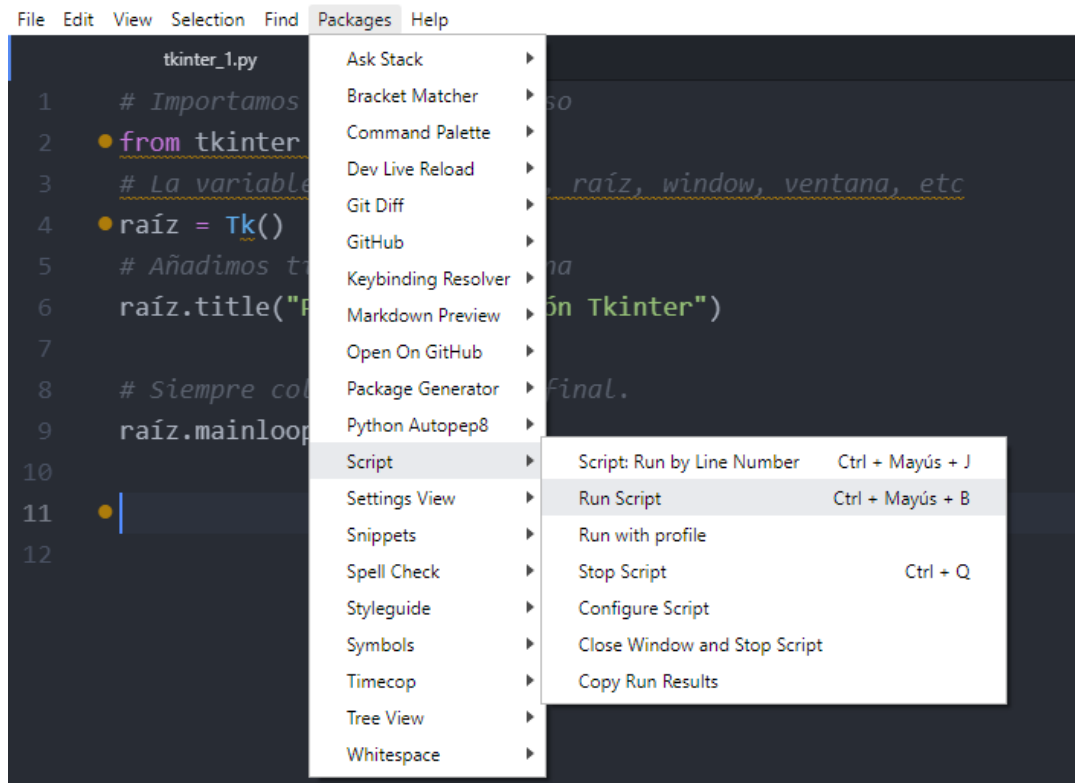


Figura 1.2: Ejecutar Script con Tkinter en ATOM.

Que nos da como resultado lo siguiente.

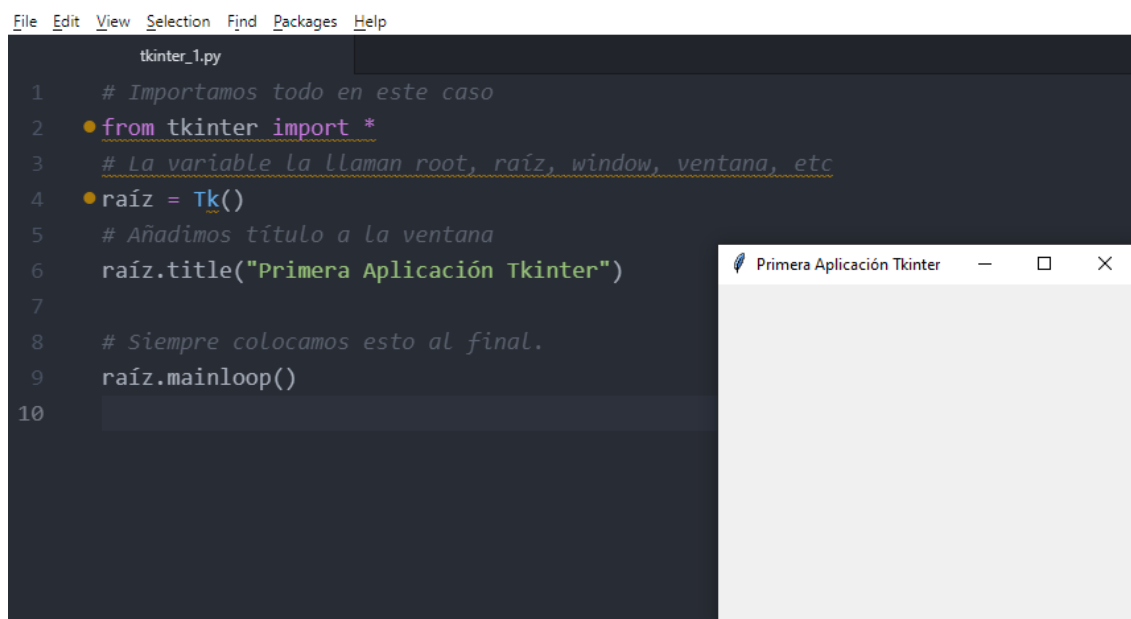


Figura 1.3: Ejemplo básico con Tkinter-Resultado

Para ello hemos arrastrado la ventana desde el límite inferior derecho, que por defecto venía más pequeña

Por el momento esa aplicación no hace nada.

Y lo único que hicimos fue cambiarle el título.

Esa ventana permanecerá abierta mientras no la cerremos en el aspa.

Veremos si podemos hacer alguna cosa más en el siguiente ejemplo.

En este caso vamos a indicarle que no se pueda ampliar la ventana, le añadiremos las dimensiones, y le cambiaremos el color de fondo.

Es conveniente recordar los diferentes nombres que suelen asignarse a la ventana, raíz, máster, etc.

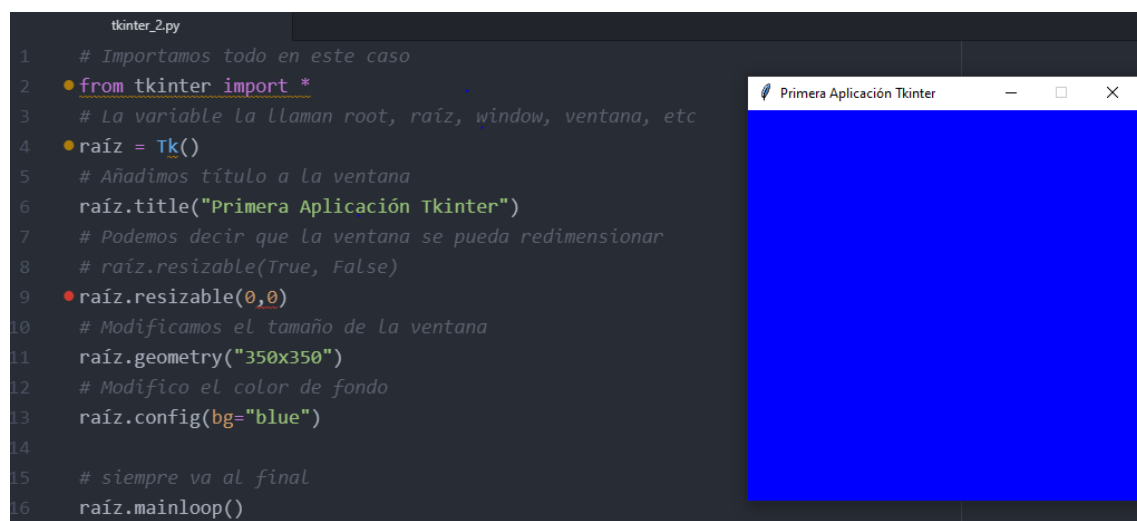


Figura 1.4: Ejemplo básico con Tkinter-cambio de color de fondo, etc.

Nota: Algunas veces aparece un color amarillo, Es un warning, y normalmente no hay que hacerlo caso. En el caso de color rojo, muchas veces, si el código ejecuta normalmente, suele tener que ver con el formato.

Veremos como queda si mejoro un poco la sintaxis de esa línea (simplemente dando un espacio más):

```
tkinter_2.py
1  # Importamos todo en este caso
2  • from tkinter import *
3  # La variable la llaman root, raíz, window, ventana, etc
4  • raíz = Tk()
5  # Añadimos título a la ventana
6  raíz.title("Primera Aplicación Tkinter")
7  # Podemos decir que la ventana se pueda redimensionar
8  # raíz.resizable(True, False)
9  raíz.resizable(0, 0)
10 # Modificamos el tamaño de la ventana
11 raíz.geometry("350x350")
12 # Modifico el color de fondo
13 raíz.config(bg="blue")
14
15 # siempre va al final
16 raíz.mainloop()
```

Figura 1.5: Ejemplo básico con Tkinter. Explicación en ATOM

En aquellos casos que el código funcione no haremos tanto caso a estas indicaciones

También podríamos hacer una simple aplicación para aprender el uso de los botones.

Lo que vamos a hacer a continuación es colocar un botón que, al hacer click sobre el mismo incremente un contador, y se imprima un mensaje.

Hemos colocado el código en el propio ATOM, la ejecución del script en la parte derecha, y en la parte inferior veremos que, cada vez que hacemos click se incrementa el contador.

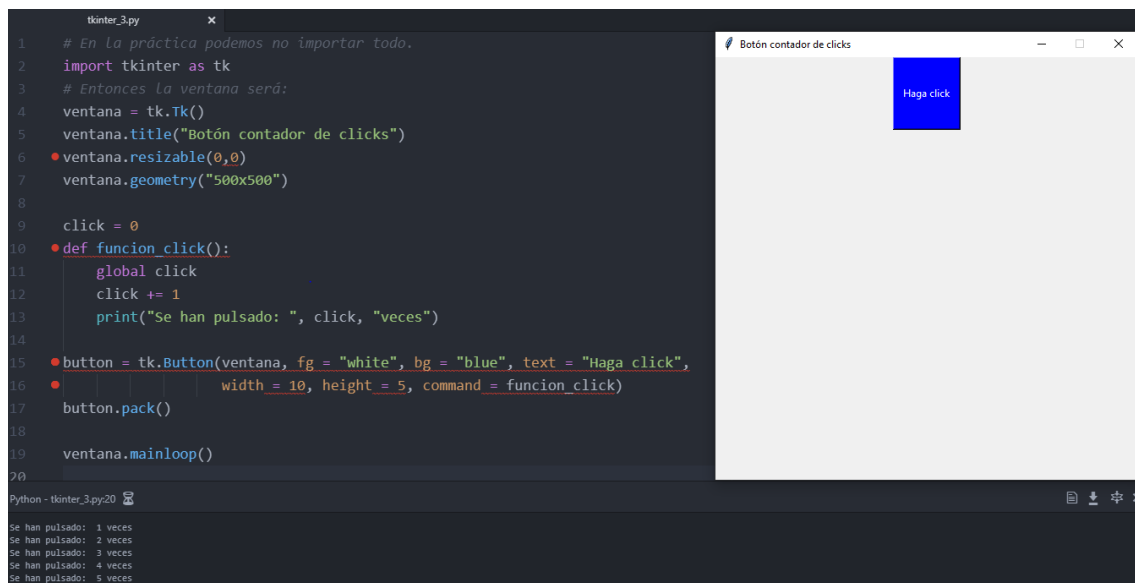


Figura 1.6: Otro Ejemplo con Tkinter.

En el siguiente ejemplo, vamos a trabajar con checkbuttons, con labels, y con algunas cosas ya vistas.

Lo llamativo será ver como ubicamos esos checkbuttons con "grid" indicándole filas (rows) y columnas (columns).

El código se hace amplio por lo cual vamos a imprimir, por una parte el código y por otra la ejecución

```

tkinter_4.py
1  import tkinter as tk # En la práctica podemos no importar todo.
2  ventana = tk.Tk() # Entonces la ventana sería así.
3  ventana.title("Checkbutton y label")
4  ventana.resizable(0,0)
5  ventana.geometry("500x500")
6
7  def funcion_opcion():
8      print("Variable 1:", v1.get(), "- Variable 2:", v2.get(), "- Variable 3:", v3.get())
9
10     # label
11     tk.Label(ventana, text = "Elija la opción que desee")
12
13     # Checkbuttons - (primero las variables que almacenan los clicks del checkbutton).
14     v1 = tk.IntVar() # variable 1
15     v2 = tk.IntVar() # variable 2
16     v3 = tk.IntVar() # variable 3
17     tk.Checkbutton(ventana, text = "Opción 1", variable = v1).grid(row=0, column=0)
18     tk.Checkbutton(ventana, text = "Opción 2", variable = v2).grid(row=0, column=1)
19     tk.Checkbutton(ventana, text = "Opción 3", variable = v3).grid(row=0, column=2)
20
21     tk.Button(ventana, text = "Mostrar las opciones elegidas",
22             fg="white", bg="blue", command = funcion_opcion).grid(row=0, column=3)
23
24     ventana.mainloop()

```

Figura 1.7: Checkbuttons en Tkinter.

Hemos hecho 2 ejemplos, siendo el segundo de ellos el indicado, en el cual le decimos a la opción 2 y a la opción 3 que SI, y NO a la opción 1, y así lo muestra la "cmd".

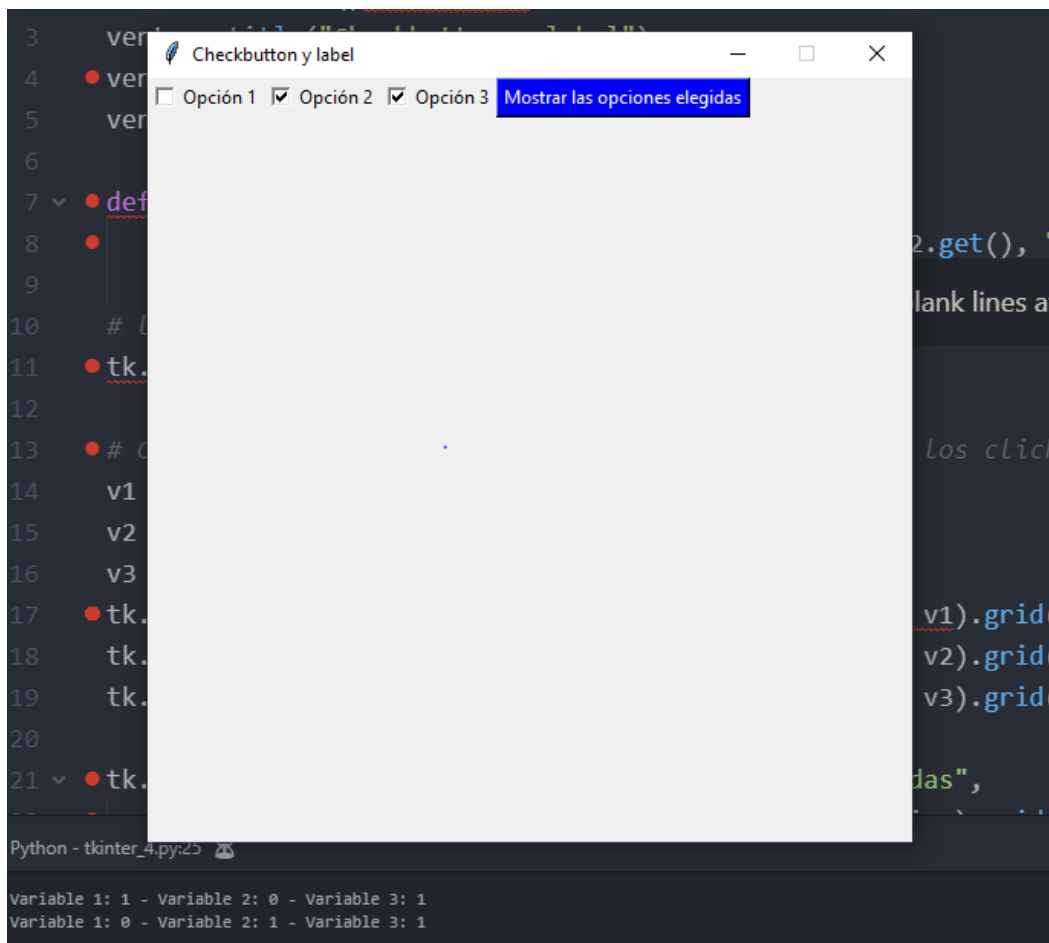


Figura 1.8: Checkbuttons, resultado

Ahora vamos a hacer una aplicación que sea capaz de recoger el Nombre y el Apellido de la persona. Al ejecutar la aplicación lo que añadimos es el nombre que indicamos: Isabel Maniega, por ejemplo. Y al hacer click sobre el botón se borraría, y aparecería en la "cmd" impreso.

Como aclaración diremos lo siguiente:

Cuando insertamos un dato le indicamos: "inserta desde la posición 0" y no le indico nada inicialmente (" ").

Las posiciones en la Aplicación podríamos cambiarlas, pero de momento lo dejaremos así para mayor simplicidad.

```
tkinter_5.py
1 import tkinter as tk
2 ventana = tk.Tk()
3
4 • def funcion_usuarios():
5     print("Nombre:", e1.get(), "- Apellido:", e2.get())
6     e1.delete(0, tk.END) # borra desde el inicio (0) hasta el final (END)
7     e2.delete(0, tk.END)
8
9 • tk.Label(ventana, text = "INGRESE NOMBRE Y APELLIDO").grid(row=0, column=0)
10 • tk.Label(ventana, text = " Nombre ").grid(row=1,column=0)
11 • tk.Label(ventana, text = " Apellido ").grid(row=2, column=0)
12 # entry
13 e1 = tk.Entry(ventana)
14 e1.insert(0, "")
15 e1.grid(row=1, column=1)
16 e2 = tk.Entry(ventana)
17 e2.insert(0, "")
18 e2.grid(row=2, column=1)
19
20 • tk.Button(ventana, text="Mostrar", fg="white", bg="blue", command=funcion_usuarios).grid(row=3,column=0)
21
22 ventana.mainloop()
```

Figura 1.9: Entry

Vemos como se ha quedado en blanco nuevamente.

Y se imprime en la "cmd"

```
21
22 ventana.mainloop()

Python - tkinter_5.py:20
Nombre: Isabel - Apellido: Maniega
```

Figura 1.10: Aplicación con Entry

Lo siguiente que vamos a hacer es crear una aplicación que sería capaz de mostrar gráficas de Matplotlib

```

tkinter_6.py x
1  import tkinter as tk
2
3  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
4  from matplotlib.backends import key_press_handler
5  from matplotlib.figure import Figure
6  import pandas as pd
7
8  ventana = tk.Tk()
9  ventana.title("Embedding in Tk")
10 df = pd.DataFrame({"x": [0,1,2,3,4], "y": [10,15,5,25,30]})
11
12 figura = Figure(figsize=(6, 4))
13 figura.add_subplot(111).plot(df.y)
14
15 canvas = FigureCanvasTkAgg(figura, ventana)
16 canvas.draw()
17 canvas.get_tk_widget().pack()
18
19 toolbar = NavigationToolbar2Tk(canvas, ventana)
20 toolbar.update()
21 canvas.get_tk_widget().pack()
22
23 tk.mainloop()

```

Figura 1.11: Matplotlib con Tkinter.

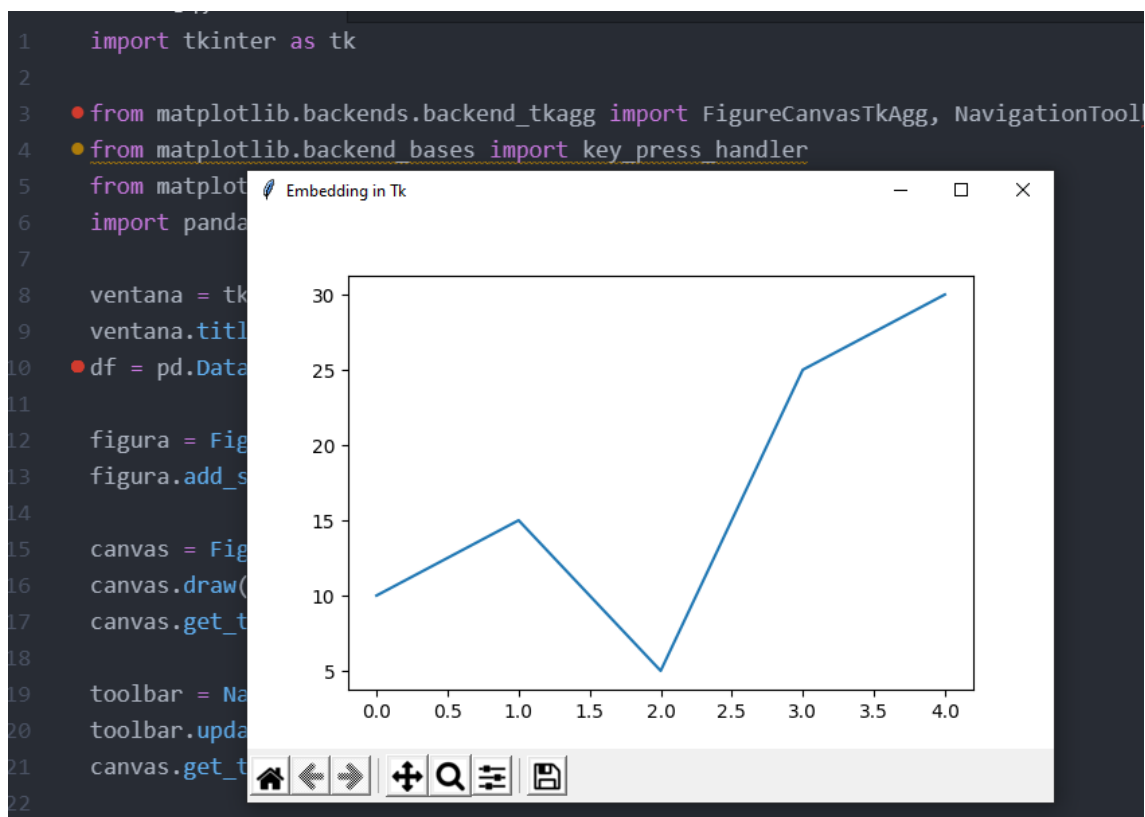


Figura 1.12: Gráficas Matplotlib en Tkinter

Podremos jugar con las gráficas de la siguiente manera, el segundo botón empezando por la derecha, es el que nos va a permitir lo siguiente:

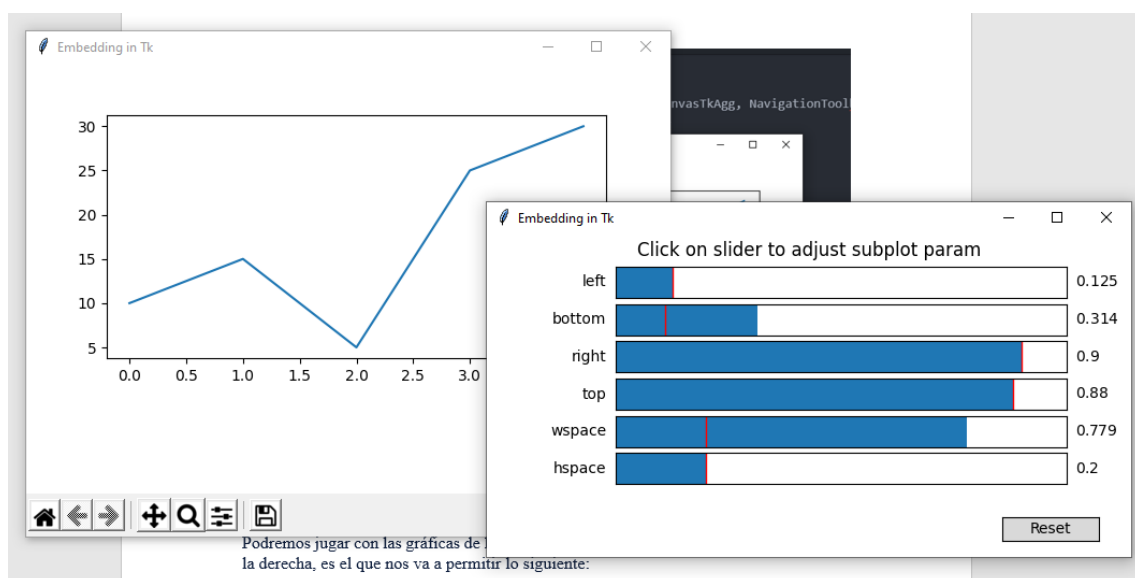


Figura 1.13: Ajustes en las gráficas Matplotlib

Es decir el ajuste de las gráficas.

Hemos dicho que para cerrar una aplicación podemos hacer click en el aspa,
También deberíamos añadir un botón con la misma finalidad.

En este caso hemos indicado que la función será “privada” (_quit)

Además hemos indicado que el botón se encuentre colocado
abajo.(BOTTOM)

```
tkinter_7.py
1  import tkinter as tk
2
3  ● ventana=tk.Tk()
4
5  ventana.title("destroy")
6  ventana.geometry("500x500")
7  ● ventana.resizable(0,0)
8
9  ● def _quit():
10     ventana.quit()
11     ventana.destroy()
12
13  ● button = tk.Button(ventana, text="Quit", command=_quit)
14     button.pack(side=tk.BOTTOM)
15
16  tk.mainloop()
```

Figura 1.14: Ventana de cierre

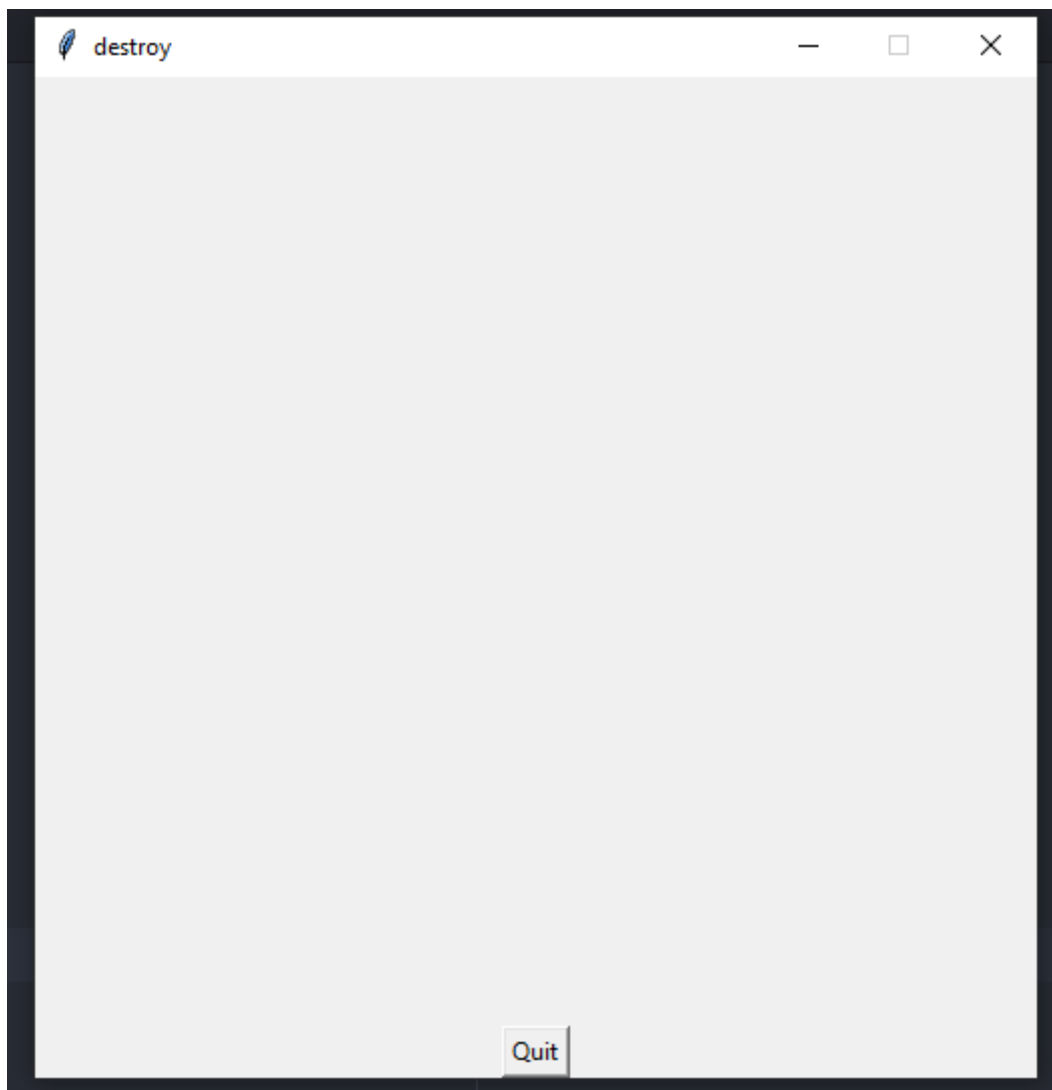


Figura 1.15: Botón de cierre

Si queremos que un Label o cualquier elemento esté situado en unas coordenadas concretas de la Aplicación simplemente le diremos `.place` y las coordenadas en `x` e `y`.

OJO: Si hemos dicho que 500x500 todo aquello que se encuentre por fuera no se verá.


```
tkinter_8.py
1 import tkinter as tk
2
3 ventana = tk.Tk()
4 ventana.title("ubicación en la pantalla")
5 ventana.geometry("500x500")
6 ventana.resizable(0, 0)
7
8 tk.Label(ventana, text="label 1", bg="blue", fg="white").place(x=20, y=20)
9 tk.Label(ventana, text="label 2", bg="blue", fg="white").place(x=200, y=200)
10 tk.Label(ventana, text="label 3", bg="blue", fg="white").place(x=400, y=400)
11 # si quisiéramos ubicarlo en otro lugar no se vería
12 tk.Label(ventana, text="label 4", bg="blue", fg="white").place(x=2000, y=2000)
13 # este no se va a ver.
14 ventana.mainloop()
```

Figura 1.16: Label

Cuyo resultado es:

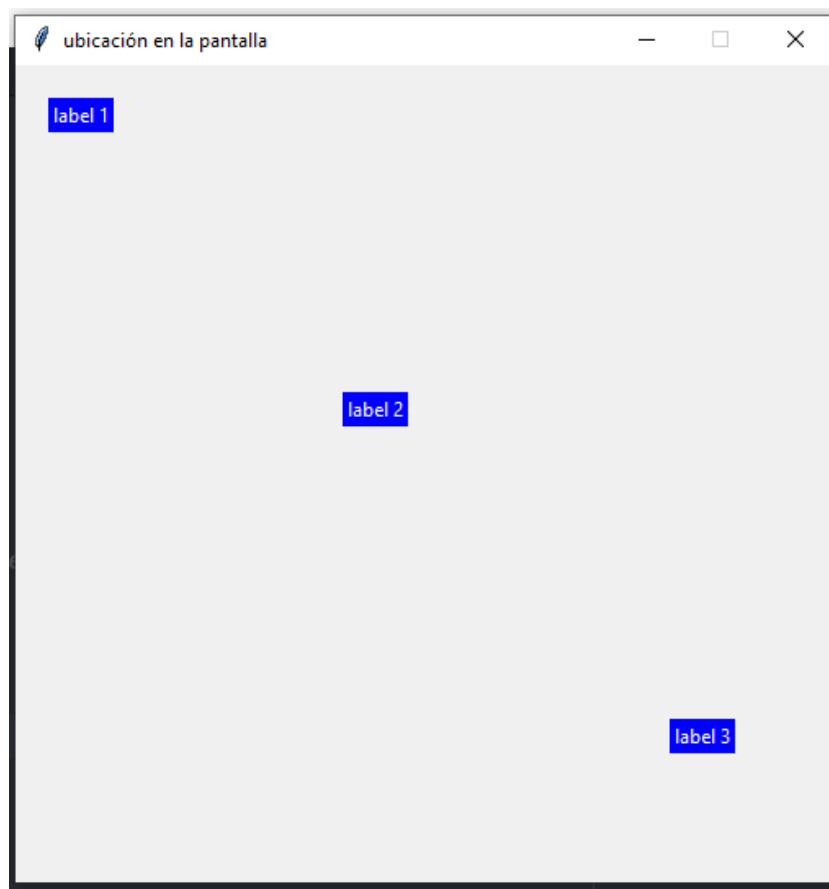


Figura 1.17: Ubicación en la ventana

2. TRANSFORMAR EN APLICACIÓN DE ESCRITORIO

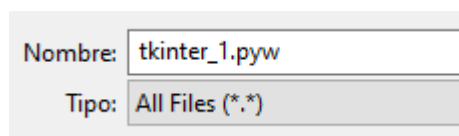
Hasta entonces hemos ejecutando la aplicación en el propio IDE.

Lo que tenemos es un .py, pero lo que queremos es una Aplicación.

Algo que podamos tener en el Escritorio o una carpeta y podamos ejecutar sin ir a la “cmd” o al propio IDE.

Para ello, lo que hacemos es lo siguiente.

Tomamos como ejemplo, cualquiera de las aplicaciones. Y nos fijamos en la extensión.



Nombre:	tkinter_1.pyw
Tipo:	All Files (*.*)

NOTA: La guardaríamos como .pyw para que funcione como una aplicación de escritorio al hacer click sobre el propio script en el escritorio.

No obstante, lo que se puede hacer es lo siguiente:

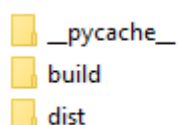
Ahora, lo siguiente será modificar nuestro script en un ejecutable.

Primero nos vamos a la cmd: “pip install pyinstaller”

Después le indicamos:

```
pyinstaller --windowed --onefile tkinter_1.py
```

Esto me generará estas carpetas:



Y en “dist” tenemos lo siguiente:

tkinter_1	16/04/2021 13:42	Aplicación	10.133 KB
-----------	------------------	------------	-----------

Una aplicación de escritorio, la cual podemos abrir con un doble click

Otra forma de hacerlo:

Nos vamos a la “cmd”: **pip install auto-py-to-exe**

Y una vez en la carpeta donde se encuentre el .py: **auto-py-to-exe tkinter_1.py**

Eso nos abre una aplicación tal que así, muy simple.

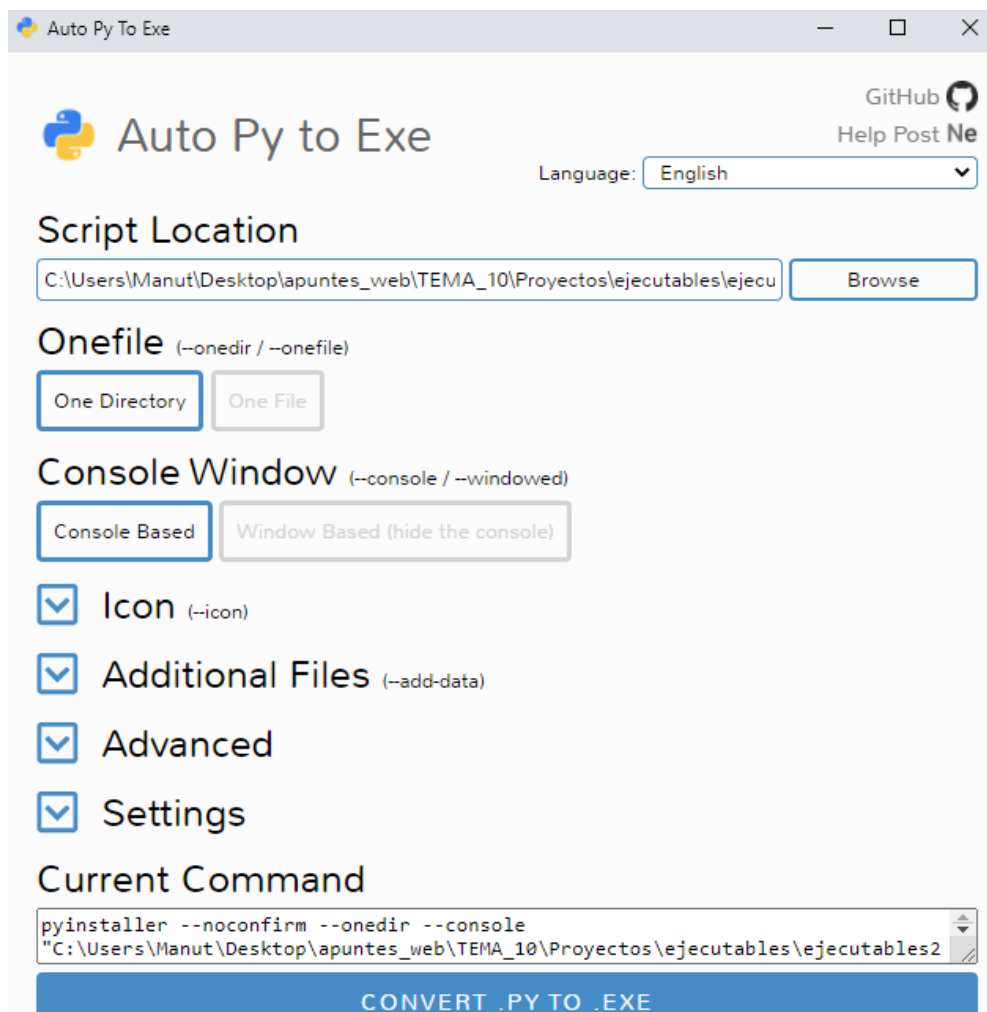


Figura 2.1: Aplicación que usa Pyinstaller

Le indicamos el .py le decimos “convert .py to .exe” y ya está.

python38.dll	16/04/2021 13:35	Extension de la ap...	4.111 KB
select	16/04/2021 13:35	Python Extension ...	27 KB
tcl86t.dll	16/04/2021 13:35	Extensión de la ap...	1.666 KB
tk86t.dll	16/04/2021 13:35	Extensión de la ap...	1.434 KB
tkinter_1	16/04/2021 14:08	Aplicación	1.992 KB
tkinter_1.exe.manifest	16/04/2021 14:08	Archivo MANIFEST	2 KB
ucrtbase.dll	16/04/2021 13:35	Extensión de la ap...	978 KB

Figura 2.2: La Aplicación

Uno de esos archivos será el que buscamos.

De modo que:

Aquí vemos como se abre de fondo algo que no queremos para la aplicación.

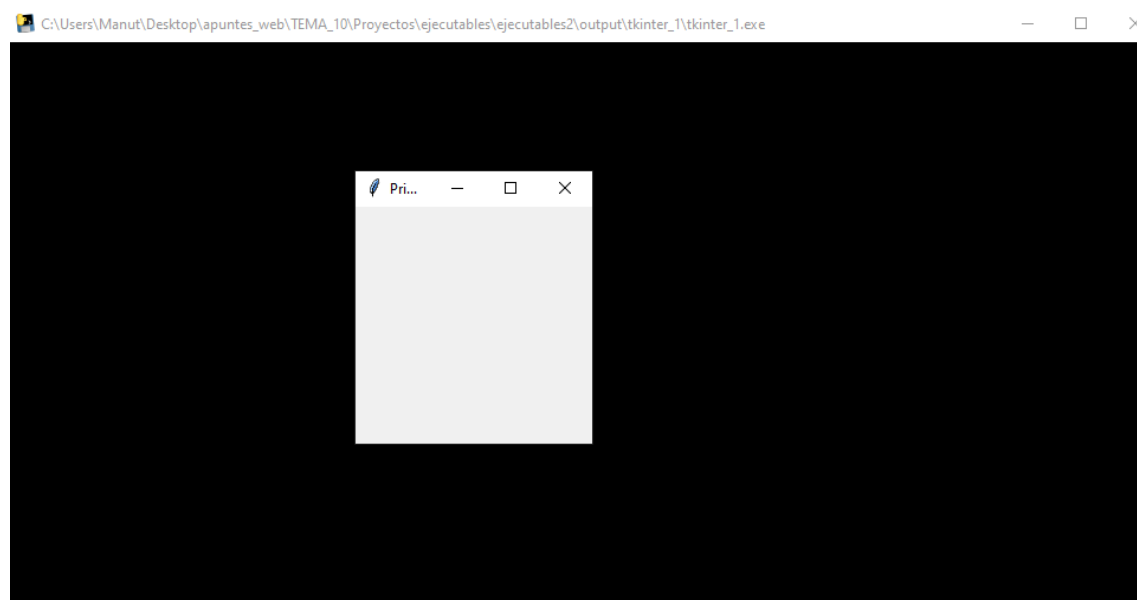


Figura 2.3: No abre realmente como una Aplicación

El objetivo sería guardarlo en .pyw, cuyo resultado sería así:

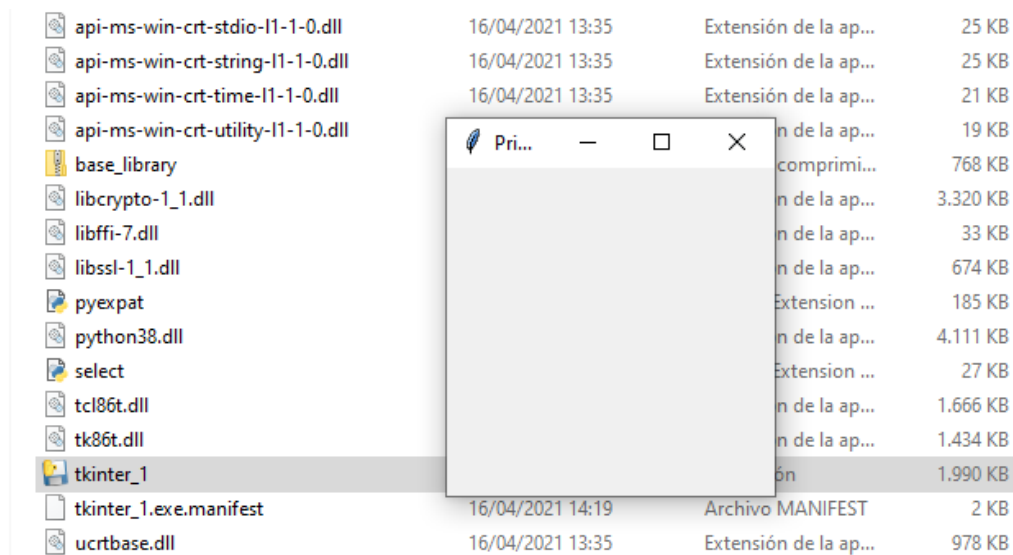
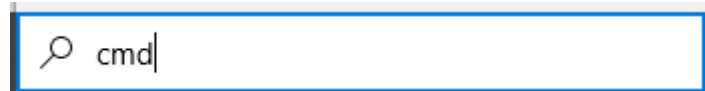


Figura 2.4: Aplicación con Tkinter

3. PYGUBU: UN “*BUILDER*” PARA TKINTER

Nos vamos al botón de búsqueda (search button) y escribimos “cmd”



Le decimos ejecutar como administrador

Nos preguntará si damos permiso, y le decimos que sí.

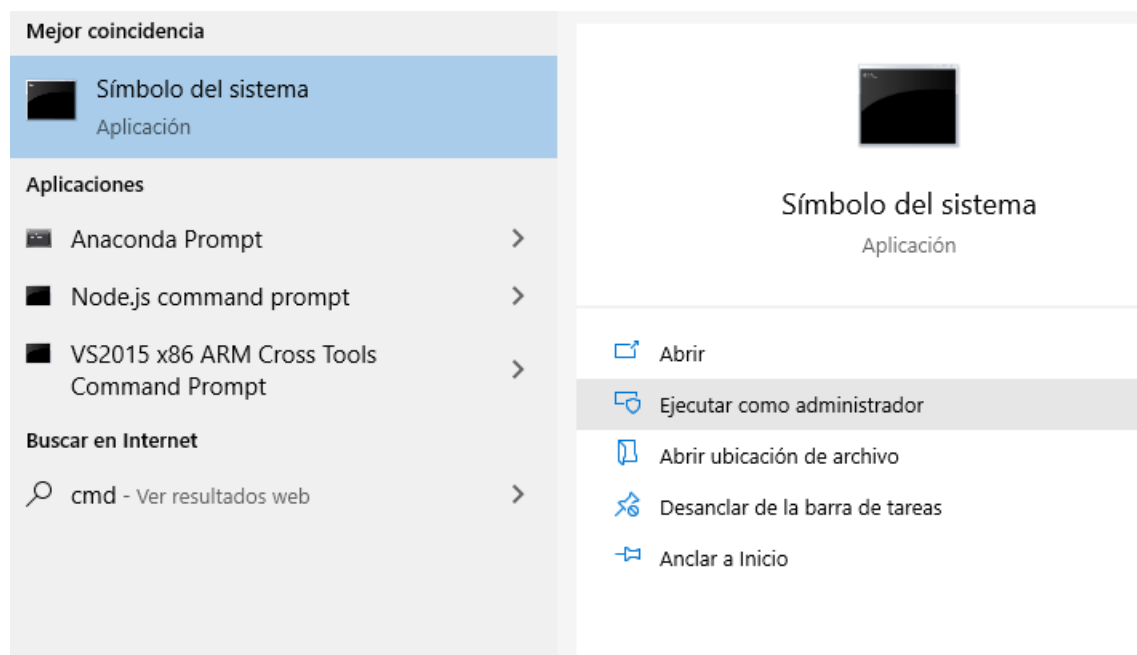
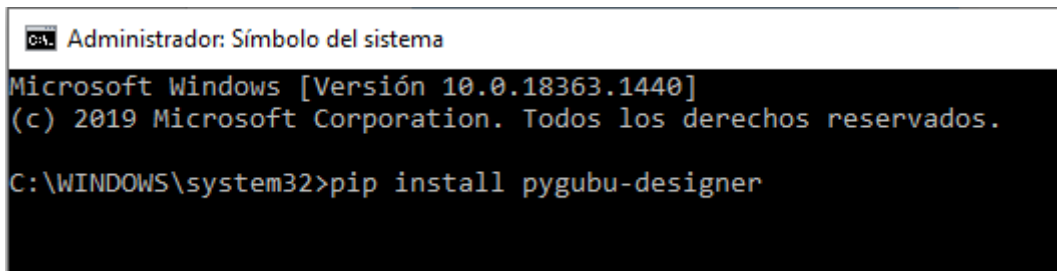


Figura 3.1: Ejecutar como administrador

En este Link encontramos información acerca de la instalación.

<https://pypi.org/project/pygubu-designer/>

Y escribimos “pip install pygubu-designer”



```
C:\> Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.18363.1440]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>pip install pygubu-designer
```

Figura 3.2: Instalación de pygubu-designer

Os dará una ruta donde se instala,

c:\users\manut\appdata\local\programs\python\python38\lib\site-packages
en mi caso.

El propio pygubu también es instalado.

(por eso elegimos esta opción y no “pip install pygubu”)

Ahora nos vamos a esa ruta de nuestro PC,

Nos vamos a “Python38” (en mi caso) y de ahí “Scripts.

En esta carpeta debería estar el archivo que queremos.

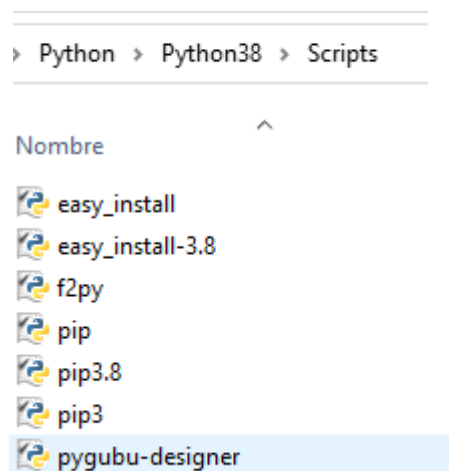


Figura 3.3: Pygubu designer

Y simplemente hacemos doble click sobre “pygubu-designer”.

Que es algo tal que así:



Figura 3.4: Pygubu designer

Y lo cual nos ahorrará tiempo para hacer el diseño.

Cosas que se pueden hacer:

Ejemplos:

Top Level

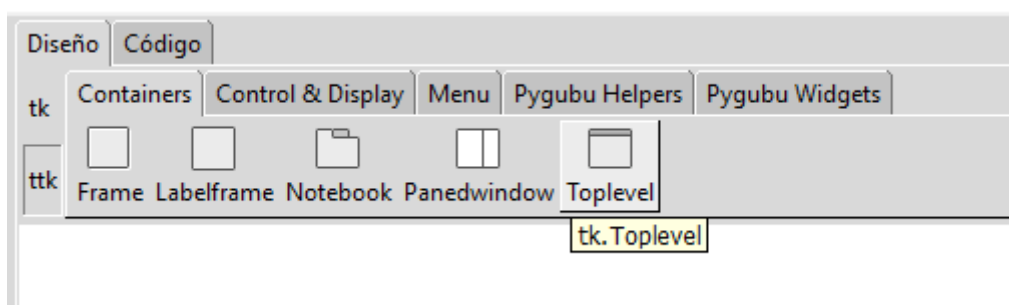


Figura 3.5: Diseño

El cual se puede expandir

Button

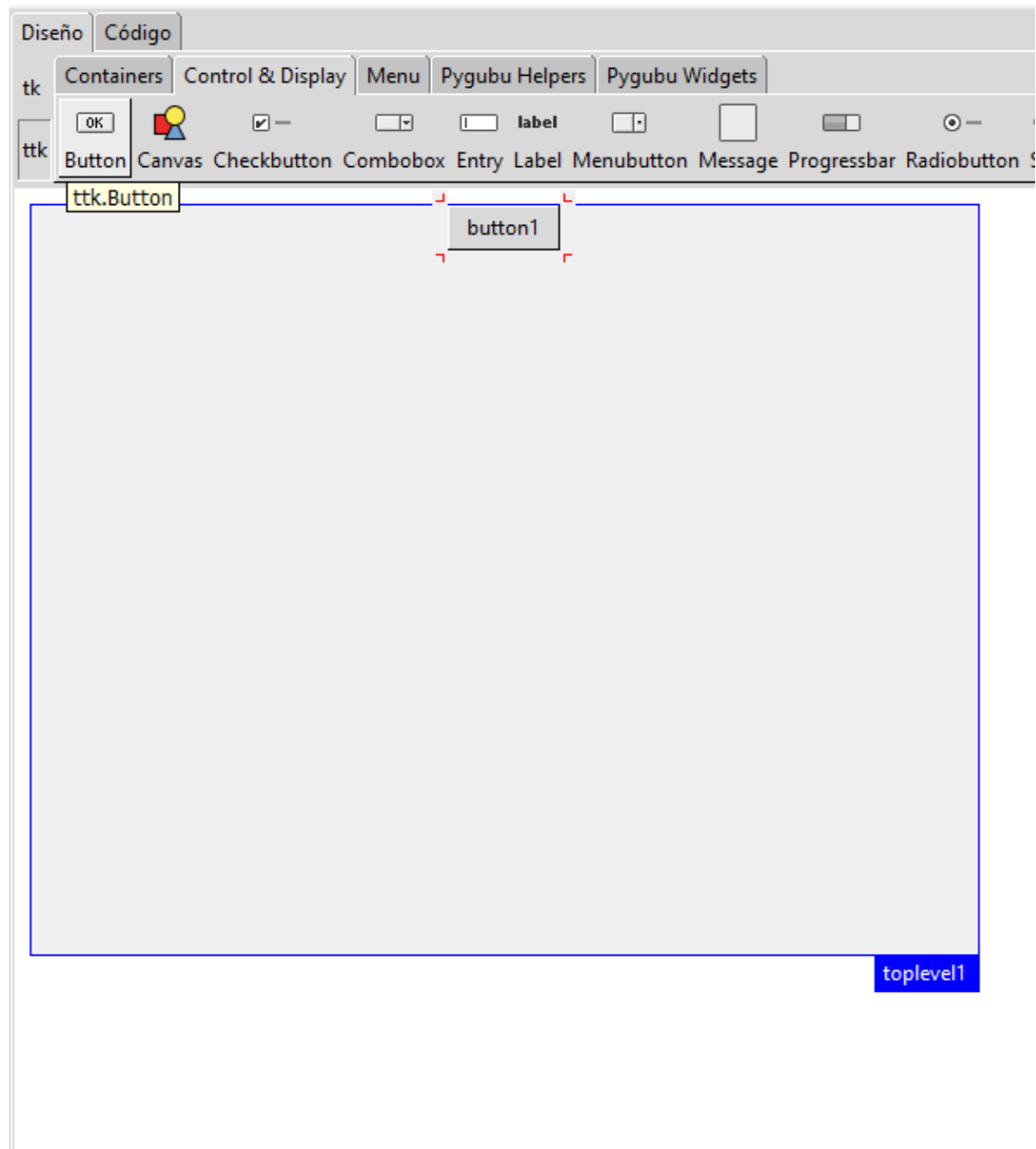


Figura 3.6: Botón en Pygubu

Label

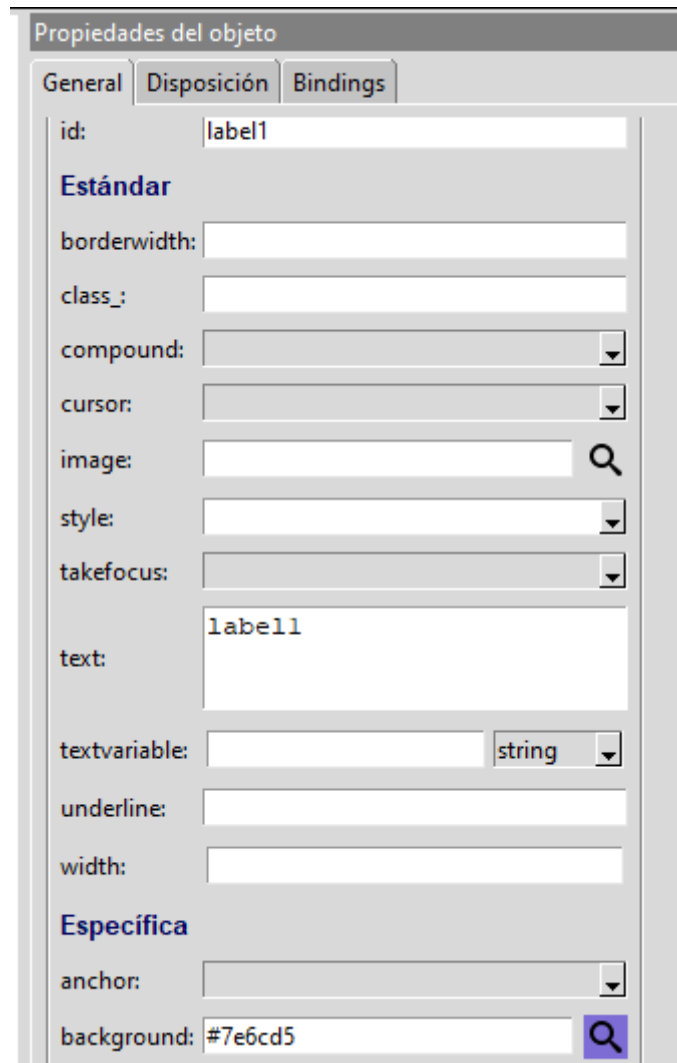


Figura 3.7: Label en Pygubu

Podemos modificar los atributos del botón o del “Label”.
Por ejemplo el color.

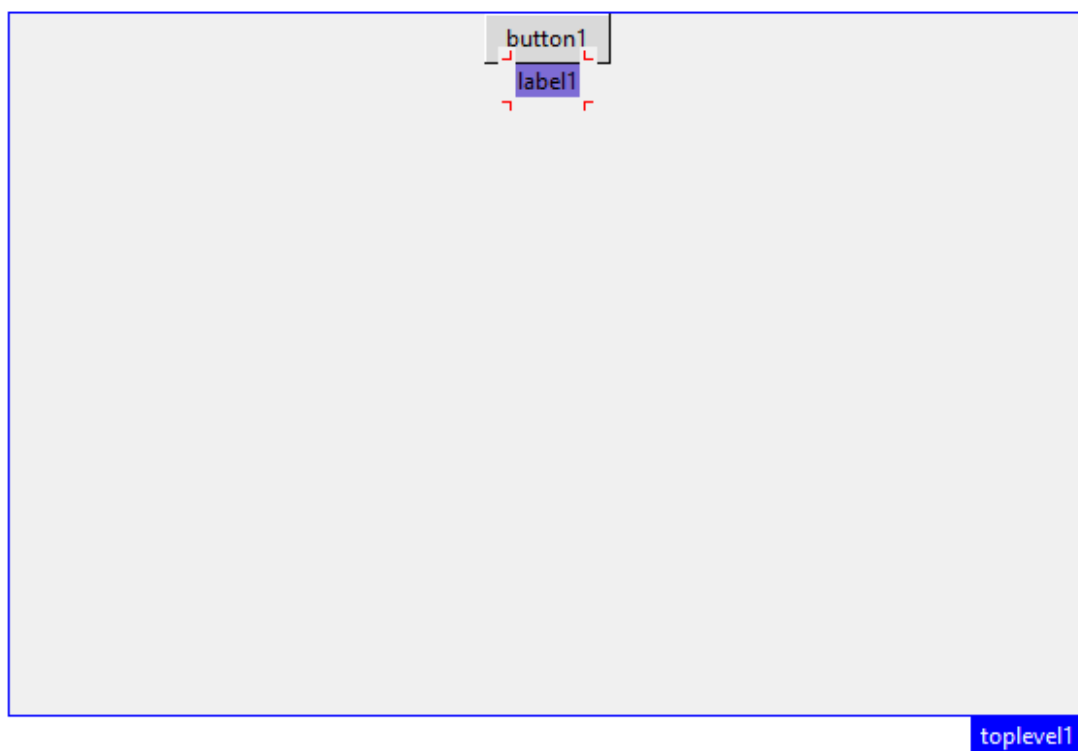


Figura 3.8: Pygubu

Para obtener el código:

Código → Generar

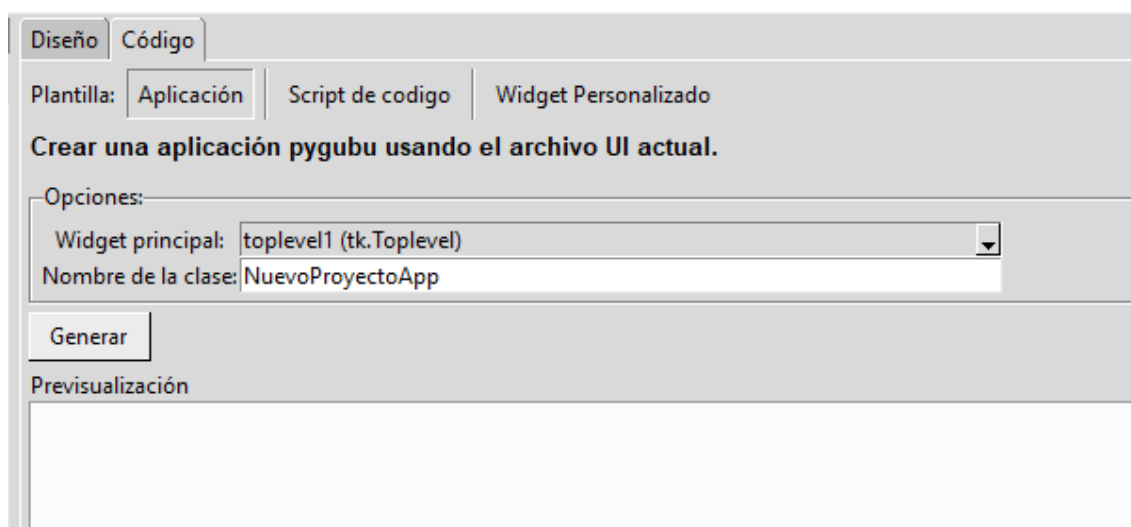


Figura 3.9: Generar código en Pygubu

Y nos generará el código en Python

Pero debemos ir a : “Script de código” y generarlo allí

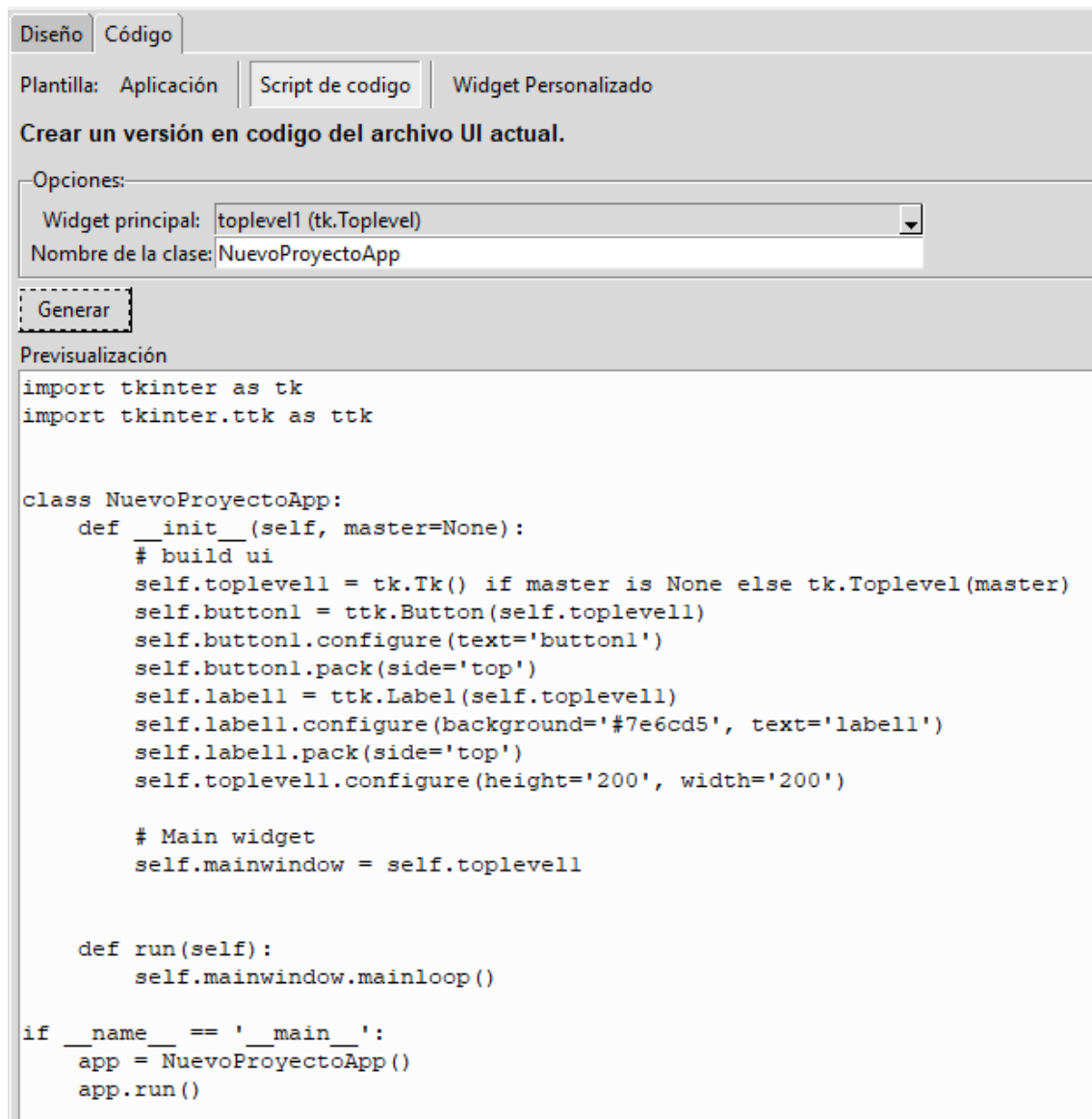


Figura 3.10: Código generado

Que es un código (como vemos Orientado a Objetos-POO) de lo que hemos hecho.

Le decimos: “guardar”

Y debemos indicarle el nombre y Tipo, así como la ubicación en nuestro PC.

Nombre:	nuevoproyectoapp	▼
Tipo:	Python Script	▼

Nos vamos al lugar donde lo hemos guardado, y tenemos:

Botón derecho → Editar con IDLE

Nos sale el mismo código

```
File Edit Format Run Options Window Help
import tkinter as tk
import tkinter.ttk as ttk

class NuevoProyectoApp:
    def __init__(self, master=None):
        # build ui
        self.toplevel1 = tk.Tk() if master is None else tk.Toplevel(master)
        self.button1 = ttk.Button(self.toplevel1)
        self.button1.configure(text='button1')
        self.button1.pack(side='top')
        self.label1 = ttk.Label(self.toplevel1)
        self.label1.configure(background='#7e6cd5', text='label1')
        self.label1.pack(side='top')
        self.toplevel1.configure(height='200', width='200')

        # Main widget
        self.mainwindow = self.toplevel1

    def run(self):
        self.mainwindow.mainloop()

if __name__ == '__main__':
    app = NuevoProyectoApp()
    app.run()
```

Figura 3.11: Código generado

Y si le damos a “Run” → Run Module (F5)

Tenemos lo mismo que hicimos:

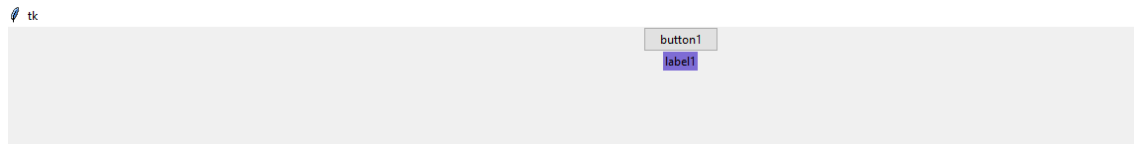


Figura 3.12: Tenemos lo que hicimos

En este caso hemos hecho una aplicación que no tiene mucho sentido, pero podríamos haber hecho algo más complejo y práctico, ubicando los botones y labels donde queramos, con mayor tamaño etc.

Existen más formas, otra es “PAGE”, el cual te genera código con el ya visto “try except” pero no lo veremos en esta ocasión.

4. PUNTOS CLAVE

- | Tkinter es una gran opción para crear aplicaciones GUI.

